INE5645 – Programação Paralela e Distribuída – Prova 1 - 02052018

ALUNO \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. (Formas de Escalonamento em Java) No código seguinte,   
     
   1. **ExecutorService** *executar\_leitor* = **Executors**.**newFixedThreadPool(4);**   
   2. **ScheduledExecutorService** *executar\_escritor* =   
    **Executors.newScheduledThreadPool(1);**   
    try {   
   3. executar\_leitor(new **Leitor**( sharedLocation ));   
   4. executar\_escritor.scheduleAtFixedRate(new   
    **escritor**(sharedLocation),  
    0,5,TimeUnit.MILLISECONDS);  
    }

a) Qual a diferença entre **ExecutorService** e **ScheduledExecutorService** (0,25)

b) Threads leitoras ou escritoras quando são criadas, de acordo com as linhas 3 e 4, passam para o estado \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ para poderem ser executadas. Assim, threads prontas no pool de threads podem ser escalonadas e processadas. (0.25)  
  
  
c) (Verdade/Falso) Threads leitoras e escritoras são códigos sequenciais, e como tal, quando estão   
 no pool e compartilham, concorrentemente, um único núcleo do processador. (0.25)

d) (Verdade/Falso) Na linha 1, **4** threads estarão na fila do pool aguardando entrada no pool de   
 tamanho 4 de threads. (0.25)  
  
(e) (Verdade/Falso) **ExecutorService** é conveniente para uso com threads escritoras, porque essas threads podem ser interrompidas quando executadas. (0.25)  
  
(f) (Verdade/Falso) **ScheduledExecutorService** é conveniente para uso com threads escritoras, porque essas threads devem ser escalonadas e executadas num tempo de processador arbitrário suficiente para processar de uma em uma thread, toda a thread de uma única vez. (0.25)

**Obs: As notas da Tarefa 1 e 2 no Moodle dependerá do número de acertos de suas respostas nesta  
prova .**

1. OpenMP – (Modelo de Programação)Indique (Verdade/Falso). Sublinhe o que encontrar de errado nas seguintes questões. (0.25 cada)  
     
   (a) (Verdade/Falso) Todos os programas OpenMP começam como uma “Thread Master”. Essa thread executa sequencialmente até a primeira definição de uma **região paralela** ser encontrada.

(b) (Verdade/Falso) Quando uma “thread Master” chega a uma definição de uma região paralela, ela cria um conjunto de “threads” que define uma **seção paralela**. A “thread Master” faz parte do conjunto de “threads” e possui o número de identificação “0”.

(c) (Verdade/Falso) Quando o conjunto de “threads” completar a execução dos comandos na região paralela, existe um ponto de sincronização implícito (“**barreira**”) no final de uma região paralela, sincronizando o fim de execução de cada “thread”. Somente a “thread” Master continua além desse ponto.

(d) (Verdade/Falso) Em uma execução OpenMP, o número de “threads” numa região paralela é determinado pelos seguintes fatores, em ordem de precedência:

1. Utilização da **função** omp\_set\_num\_threads() no código C/C++;
2. Definindo a **variável de ambiente** OMP\_NUM\_THREADS = ... , antes da execução;

3. Implementação **padrão do ambiente**: número de **núcleos** processadores, se nada é dito de   
 1. e 2.

1. OpenMP – Indique Verdade ou Falso e sublinhe o que for falso. (0.25 cada)
2. (Verdade/Falso) Suporta um modelo de paralelismo de dados.
3. (Verdade/Falso) Combina código serial e paralelo em um único código fonte.
4. (Verdade/Falso) Combina código paralelo e serial em um único código fonte.
5. (Verdade/Falso) Todo o paralelismo é construído a partir de Diretivas de Linkedição.
6. Considere os códigos:   
   (a) (Verdade/Falso) Os dados globais de um programa OpenMP, são compartilhados entre as threads. (0.25)  
   (b) Estes dois segmentos de código não são equivalentes: (0.25)

|  |  |
| --- | --- |
| #pragma omp parallel  {  #pragma omp for   for (i=0; i< MAX; i++)   {   res[i] = huge()   }  } | #pragma omp parallel for     for (i=0; i< MAX; i++)   {   res[i] = huge();   } |

1. (a) Nos códigos abaixo, explique, brevemente, nas linhas em branco seguintes, o que significa as cláusulas **private(x,y)** e **shared(sum)**

|  |  |
| --- | --- |
| void\* work(float\* c, int N)  {   float x, y; int i;   #pragma omp parallel for **private(x,y)**   for(i=0; i<N; i++)  {  x = a[i]; y = b[i];  c[i] = x + y; | float dot\_prod(float\* a, float\* b, int N)  {   float sum = 0.0;   #pragma omp parallel for **shared(sum)**    for(int i=0; i<N; i++) {  sum += a[i] \* b[i];  }  return sum;  } |

(a) **private(x,y)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(b) **shared(sum)**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. No primeiro código seguinte, o que está errado?

|  |  |
| --- | --- |
| float dot\_prod(float\* a, float\* b, int N)  {   float sum = 0.0;   #pragma omp parallel for shared(sum)   for(int i=0; ; i<N; i++) {  sum += a[i] \* b[i];  }  return sum; } | RESPOSTA: |
| float dot\_prod(float\* a, float\* b, int N)  {   float sum = 0.0;   #pragma omp parallel for shared(sum)  for(int i=0; ; i<N; i++) {  **// Corrija o código aqui no espaço abaixo**     sum += a[i] \* b[i];  }  return sum;  } | |

1. O código abaixo é sobre o uso de **reduction(...)**, responda (Verdade/Falso).

|  |  |
| --- | --- |
| #pragma omp parallel for **reduction** (+: **sum**)  for(i=0; i<N; i++) {  **sum** += a[i] \* b[i];  } | 1. (Verdade/Falso) Existe uma cópia local de **sum** para cada thread. 2. (Verdade/Falso) Todas as cópias locais de sum são adicionadas e armazenadas na sua variável “global” |

1. Com relação a cláusula **schedule**, o que ocorre no código abaixo:

|  |  |
| --- | --- |
| O código abaixo gera e testa números primos de valor inicial start até o valor final end.  A cláusula **schedule** afeta como as iterações do *loop* são mapeadas nas threads  #pragma omp parallel for **schedule** (static, **8**)   for( int i = start; i <= end; i += 2 )   {   if ( TestForPrime(i) ) gPrimesFound++;   } | RESPOSTA:  Se start = 3, então o primeiro chunck em um número total de **8**, corresponderá as **iterações i** iguais a:   **Iterações** **i =** {....... ....... ....... ....... ......... ........ ......... .......}  (cada iteração gerará um número que será testado se é primo). |

1. No código abaixo sobre **sections** paralelas, qual a diferença quanto a regiões paralelas ?

|  |  |
| --- | --- |
| #pragma omp parallel sections  {   #pragma omp section  phase1();   #pragma omp section   phase2();   #pragma omp section   phase3();  } | RESPOSTA: |

1. Sobre os modelos de paralelismo em OpenCL:  
     
   a) **O que significa a seguinte figura em termos computacionais de modelos de paralelismo, envolvendo instrução e dados ?** (0.25)

|  |  |
| --- | --- |
|  | RESPOSTA: |

b) **O que significa a seguinte figura em termos computacionais de modelo de paralelismo, envolvendo instruções e dados ?**  (0.25)

|  |  |
| --- | --- |
|  | RESPOSTA: |

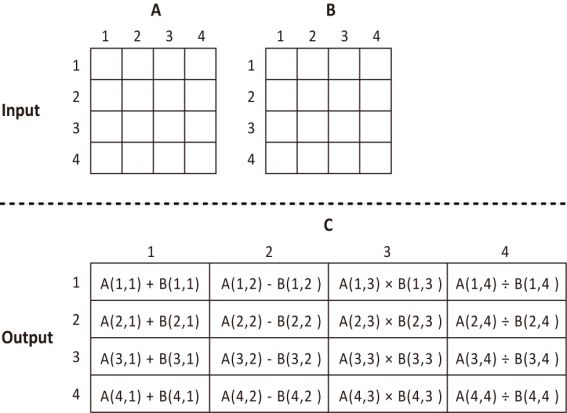
c) **O que está ilustrado nas figuras seguintes esquerda e direita, em termos da execução de códigos A() e B() paralelizados ?** (0.25)

|  |  |
| --- | --- |
| https://www.fixstars.com/images/openclbook/dtp_462724_USER_CONTENT_0_html_m613113e9.jpg | https://www.fixstars.com/images/openclbook/dtp_462724_USER_CONTENT_0_html_74632b18.jpg |

Figura à esquerda: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Figura à direita: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

d) **Considere o problema apresentado na seguinte figura:** (0.25)



Relativo às instruções do código abaixo, o que se pode afirmar sobre o modelo de paralelismo empregado ?  
  
1. \_\_kernel void dataParallel(\_\_global float\* A, \_\_global float\* B, \_\_globa l float\* C)

2. {

3. int base = 4\*get\_global\_id(0);

4. C[base+0] = A[base+0] + B[base+0];

5. C[base+1] = A[base+1] - B[base+1];

6. C[base+2] = A[base+2] \* B[base+2];

7. C[base+3] = A[base+3] / B[base+3];  
8. }